Computer Science Department

TECHNICAL REPORT

Twists, Turns, Cascades, Deque Conjecture, and Scanning Theorem

R. Sundar

Technical Report 427

January 1989

NYU COMPSCI TR-427
Sundar, Rajamani
Twists, turns, cascades,
deque conjecture, and
scanning theorem. c.2

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012

Twists, Turns, Cascades, Deque Conjecture, and Scanning Theorem

R. Sundar

Technical Report 427

January 1989

Twists, Turns, Cascades, Deque Conjecture, and Scanning Theorem

RAJAMANI SUNDAR[0]
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University

Abstract

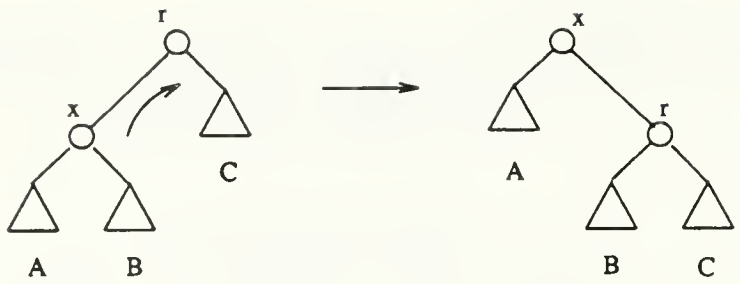
Sleator and Tarjan devised a self-adjusting binary search tree, called the *splay tree*, that is competitive with many of the balanced search tree schemes for maintaining a dictionary. They conjectured that the splay algorithm is optimal, in an asymptotic sense, among the class of self-adjusting search algorithms that use only rotations to reorganize the tree. Tarjan proved a corollary of this conjecture, called the *scanning theorem*, which states that splaying requires $O(n)$ time to access the nodes of an arbitrary n-node binary tree in symmetric order. Generalizing the scanning theorem, he conjectured that the cost of performing a sequence of m deque operations on an arbitrary n-node binary tree using splaying is $O(m+n)$. He showed that the conjecture holds when the output operations are restricted to one end of the deque. Lucas proved that the cost of executing any sequence of deque output operations on an initial tree having a certain special shape using splaying is $O(n\bar{\alpha}(n))$, where $\bar{\alpha}$ is an inverse of the Ackerman function. A *right 2-turn* is a kind of double rotation performed on a binary tree. Sleator conjectured that the number of right 2-turns in any sequence of right 2-turns and right rotations performed on an arbitrary n-node binary tree is $O(n)$. This conjecture is closely related to Tarjan's deque conjecture.

In this paper, we prove the following results:

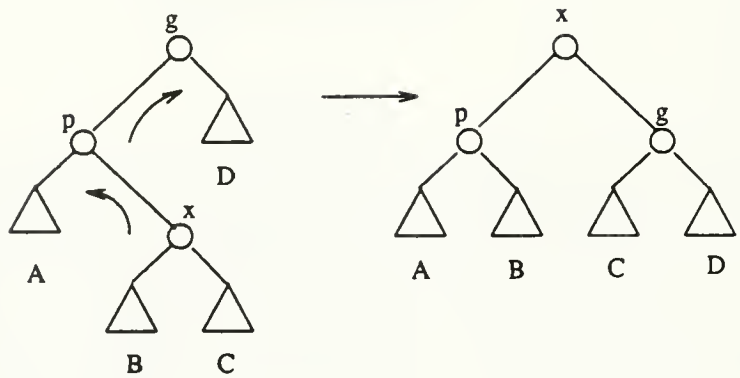
1. Nearly tight lower and upper bounds on the maximum number of each type of rotational operation in intermixed sequences of various rotational operations performed on binary trees. In particular, our lower bound for right 2-turns refutes Sleator's turn conjecture.
2. An $O((m+n)\bar{\alpha}(m+n))$ bound for the deque conjecture. This bound is derived using our upper bounds for sequences of rotational operations.
3. A generalized version of the scanning theorem.

[0]This work was supported in part by NSF grants CCR-8702271 and DMS-8703458 and ONR grant N00014-85-K-0046.

Case 1.



case 2.



Case 3.

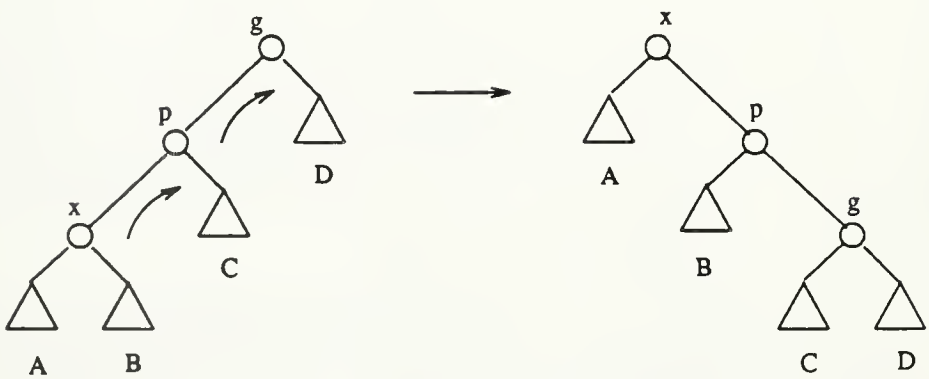


Figure 1: Splay step.

1 Introduction

A *self-adjusting binary search tree* [5] is a binary search tree which can be restructured between accesses of items by performing rotations anywhere in the tree. The *splay tree* is a self-adjusting binary search tree, devised by Sleator and Tarjan [5], which accesses items and simultaneously restructures the tree through an operation called *splay*. A splay at a node x repeats the following step until x becomes the root (See Fig. 1):

> **Splay step.**
> Let p and g denote the parent and the grandparent of x respectively.
> **Case 1.** p is the root: Make x the new root, by rotating the edge $[x, p]$.
> **Case 2.** $[x, p]$ is a left edge (*i.e.* an edge to a left child) and $[p, g]$ is a right edge, or vice versa: Rotate $[x, p]$, making g the new parent of x; Rotate $[x, g]$.
> **Case 3.** Either both $[x, p]$ and $[p, g]$ are left edges, or both are right edges: Rotate $[p, g]$; Rotate $[x, p]$.

They proved that, in an amortized sense and up to a constant factor, the splay tree is as efficient as many of the balanced search trees and the static optimal binary search tree for processing a sequence of dictionary operations. They and Tarjan [7] made the following conjectures concerning the performance of the splay tree:

Conjecture 1 (Dynamic optimality conjecture [5]) *Consider an arbitrary sequence of m searches of items in an n-node binary search tree. Let T_{SA} = the minimum time required to execute this sequence of searches on any self-adjusting binary search tree, assuming that each search costs (1+the accessed item's depth) and that each rotation costs 1. Then, the cost of performing this sequence of accesses on the splay tree is $O(n + T_{SA})$.*

Conjecture 2 (Dynamic finger conjecture [5]) *Consider an arbitrary sequence of m searches of items in an n-node binary search tree. Denote the item accessed by the j^{th} search by its symmetric order number i_j. The cost of executing this sequence of accesses using the splay algorithm is $O(m + n + \sum_{j=1}^{m-1} \log(|i_{j+1} - i_j| + 1))$.*

Conjecture 3 (Traversal conjecture [5]) *Consider accessing the items of an arbitrary, fixed n-node binary search tree T in preorder. The splay algorithm requires $O(n)$ time to execute this sequence of accesses on any binary search tree that has the same set of items as T.*

Conjecture 4 (Deque conjecture [7]) *Consider the representation of a deque by a binary tree in which the i^{th} node of the binary tree in symmetric order corresponds to the i^{th} element of the deque. The splay algorithm is used to perform deque operations on the binary tree as follows (See Fig. 2):* POP *splays at the smallest node of the tree and removes it from the tree;* PUSH *makes the inserted item the new root, with null left child and the old root as right child;* EJECT *and* INJECT *are symmetric. The cost of performing any sequence of m deque operations on an arbitrary n-node binary tree using splaying is $O(m + n)$.*

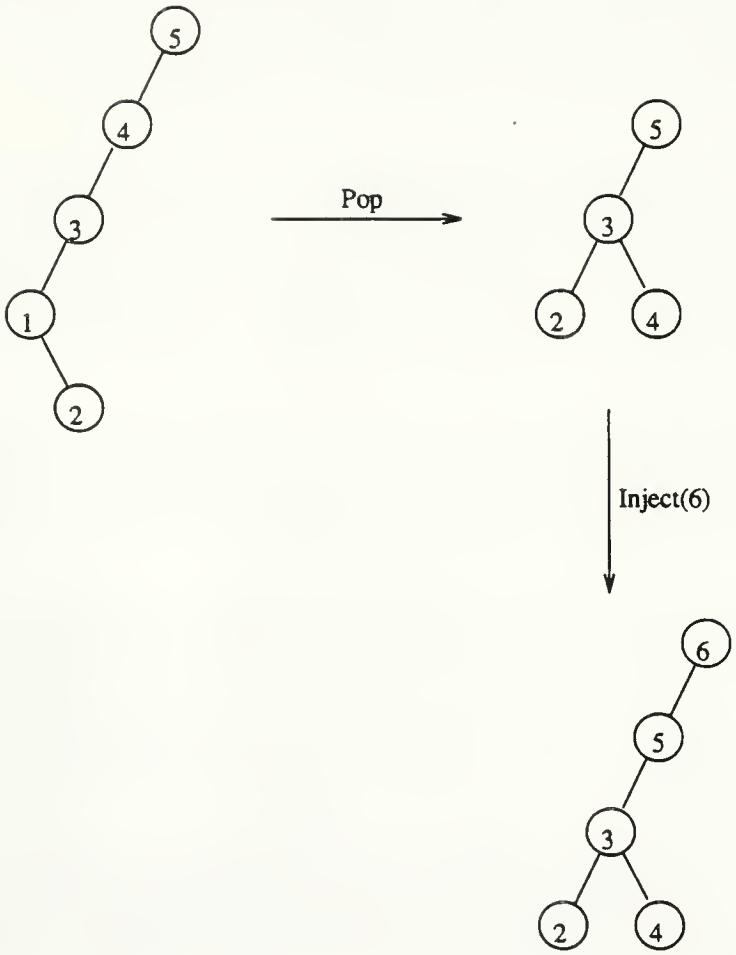


Figure 2: Deque operations.

Conjecture 5 (Turn conjecture [7]) *A* right 2-turn *on a binary tree is a sequence of two right rotations performed on the tree in which the bottom node of the first rotation is identical to the top node of the second rotation. The total number of right 2-turns in any sequence of right 2-turns and right rotations performed on an arbitrary n-node binary tree is* $O(n)$.

The dynamic optimality conjecture implies the dynamic finger conjecture and the traversal conjecture. Lucas [4] generalized the optimality conjecture to include join and split operations. The generalized version of the optimality conjecture implies the deque conjecture. The turn conjecture also implies the deque conjecture. (This fact was not known before; it is proved in the last section of this paper.)

At this point, we digress to introduce the Ackerman hierarchy of functions $\{A_i | i \geq 0\}$, its inverse hierarchy $\{\hat{\alpha}_i | i \geq 0\}$, and inverse functions $\bar{\alpha}$ and α of the Ackerman function. Define:

$$
\begin{aligned}
A_0(j) &= 2j \quad \text{for all } j \geq 1 \\
A_1(j) &= 2^j \quad \text{for all } j \geq 1 \\
A_i(j) &= \begin{cases} A_{i-1}(2) & \text{if } i \geq 2 \text{ and } j = 1 \\ A_{i-1}(A_i(j-1)) & \text{if } i \geq 2 \text{ and } j \geq 2 \end{cases} \\
\hat{\alpha}_i(n) &= \min\{k \geq 1 | A_i(k) \geq n\} \quad \text{for all } n \geq 1 \\
\bar{\alpha}(n) &= \min\{k \geq 1 | A_k(1) \geq n\} \quad \text{for all } n \geq 1 \\
\alpha(m,n) &= \min\{k \geq 1 | A_k(\lfloor m/n \rfloor) > \log n\} \quad \text{for all } m \geq n \geq 1
\end{aligned}
$$

The following table concretizes this definition:

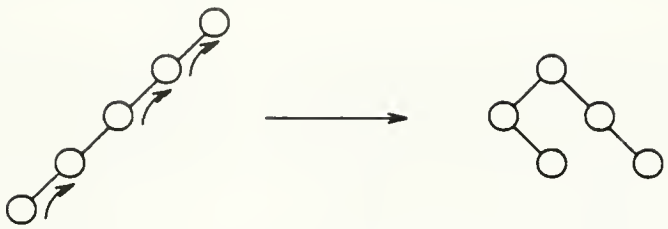
i	0	1	2	3	4
$A_i(j)$	$2j$	2^j	$2^{\cdot^{\cdot^{2}}}\}j$		
$\hat{\alpha}_i(n)$	$\lceil n/2 \rceil$	$\lceil \log n \rceil$	$\leq \log^* n$	$\leq \log^{**} n$	$\leq \log^{***} n$
$\{n \mid \bar{\alpha}(n) = i\}$		$[1,2]$	$[3,4]$	$[5,16]$	$[17, 2^{\cdot^{\cdot^{2}}}\}16]$

Next, we review previous research concerning the splay tree conjectures. Tarjan [7] proved the following theorem which is a corollary of all five conjectures:

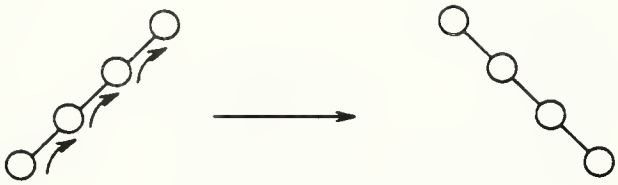
Scanning theorem [7] *Splaying requires* $O(n)$ *time to access the nodes of an arbitrary n-node binary tree in symmetric order.*

He also extended his proof and showed that the deque conjecture is true if the output operations are restricted to one end of the deque. Lucas [4] obtained an $O(n\bar{\alpha}(n))$ bound for the deque conjecture when only output operations are permitted and the initial tree is a simple path between the minimum and maximum nodes. In addition to these direct efforts towards resolving the optimality conjecture, there have been some other related works.

i. A right 3-twist



ii. A right 3-turn



iii. A right 3-cascade

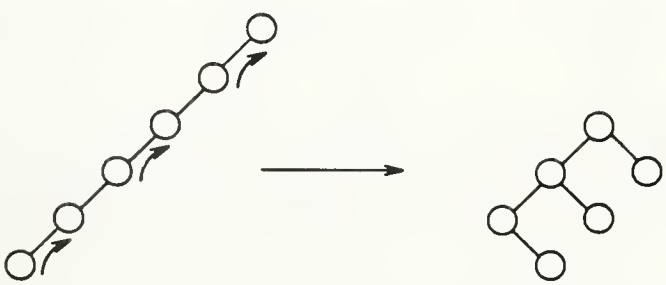


Figure 3: Twist, Turn and Cascade.

Wilber [8] gave two techniques for establishing lower bounds on the cost of performing a sequence of accesses on a self-adjusting binary search tree. Culik and Wood [2] showed that any binary tree can be transformed into any other binary tree by performing at most $2n-2$ rotations. Sleator et al. [6] improved this bound to $2n-6$, for $n > 12$, and proved that it is tight for infinitely many n.

In this paper, we perform a combinatorial analysis of sequences of various rotational operations on binary trees and thereby derive an $O((m+n)\bar{\alpha}(m+n))$ bound for the deque conjecture. This approach is very different from that of Lucas [4], which exploits the similarity between splaying and certain set union algorithms. Our approach seems more general for it is able to handle the complete set of deque operations and arbitrary initial trees. We study the following binary tree operations (See Fig. 3):

Right twist: For any $k \geq 1$, a right k-twist arbitrarily selects k different edges from a left subpath[1] of the binary tree and rotates the edges one after another in top-to-bottom order.

Right turn: For any $k \geq 1$, a right k-turn is a right k-twist that converts a left subpath of k edges in the binary tree into a right subpath by rotating the edges of the subpath in top-to-bottom order.

Right cascade: For any $k \geq 1$, a right k-cascade is a right k-twist that rotates every other edge lying on a left subpath of $2k-1$ edges in the binary tree.

A *right twist sequence* (abbreviated *RT-sequence*) is a sequence of right twists performed on a binary tree. Define $U_k(n)$, $C_k(n)$ and $T_k(n)$, respectively, to be the maximum number of k-turns, k-cascades and k-twists in any RT-sequence performed on an n-node binary tree. These numbers are well defined since, after $O(n^2)$ right rotations, the tree is transformed into a right path and no further right rotations can be performed. We derive the following almost tight bounds for $U_k(n)$, $C_k(n)$ and $T_k(n)$:

	Upper bound	Lower bound
$U_k(n)$	$O(n\hat{\alpha}_{\lfloor k/2 \rfloor}(n))$ if $k \neq 3$	$\Omega(n\hat{\alpha}_{\lfloor k/2 \rfloor}(n)) - O(n)$ if $k \neq 3$
$C_k(n)$	$O(n \log\log n)$ if $k = 3$	$\Omega(n \log\log n)$ if $k = 3$
$T_k(n)$	$O(kn^{1+1/k})$	$\Omega(n^{1+1/k}) - O(n)$

The bounds for $U_k(n)$ and $C_k(n)$ are tight if $k \leq 2\bar{\alpha}(n) - 5$ and the bounds for $T_k(n)$ are nearly tight. Our study of turns was motivated by the turn conjecture. Our bound of $\theta(n \log n)$ for $U_2(n)$ shows that the conjecture is false[2]. This is unfortunate since the turn conjecture implies the deque conjecture. The reason behind studying cascades is that a deque output operation precisely involves deleting the minimum or the maximum node

[1] A *left subpath* of a binary tree is a path in the binary tree comprising only left edges.

[2] After proving these results, the author learnt from R.E.Tarjan (personal communication) that S.R.Kosaraju has independently proved that $U_2(n) = \theta(n \log n)$. Kosaraju's lower bound construction is identical to ours, but his upper bound construction is quite different.

and performing a cascade spanning the entire left path[3] or the entire right path of the binary tree. We extend the upper bound for $C_k(n)$ to the inverse Ackerman bound for the deque conjecture. Twists are a natural generalization of turns and cascades and are worth investigating purely for their combinatorial attraction.

Our next contribution is a generalization of the scanning theorem. Define the *left depth* of a node in a binary tree to be the number of left edges on the access path of the node. For any positive integer k and real number d, such that $1 \leq d \leq n$, call a right k-twist *d-shallow* if the lowest node involved in the twist has a left depth of at most dk. Let $S^{(d)}(n)$ denote the maximum number of rotations performed by d-shallow right twists in any RT-sequence executed on an n-node binary tree. We prove that $S^{(d)}(n) = O(dn)$. It is easy to show a matching lower bound of $\Omega(dn)$ for $S^{(d)}(n)$. The upper bound for $S^{(d)}(n)$ generalizes the scanning theorem because, when the items of a binary tree are accessed in symmetric order using splaying, the splay operation accessing any item performs essentially a 2-shallow right twist (actually a right cascade) that spans the access path of the item.

The paper is organized as follows: In Section 2, we prove the bounds for $U_k(n)$ and $C_k(n)$. The upper and lower bounds for $T_k(n)$ are derived in Section 3. The inverse Ackerman bound for the deque conjecture is established in the following section. We prove the upper bound for $S^{(d)}(n)$ in Section 5. The last section poses a conjecture involving right 2-turns that implies the deque conjecture.

2 Bounds for turns and cascades

2.1 Upper bounds

We begin by introducing the notion of blocks in binary trees that will be used extensively in all our proofs. This concept is not new; it has already been used by many people, including Tarjan [7], Wilber [8], Lucas [4], and Cole et al. [1], in analyzing the effects of rotations in binary trees. Consider an n-node binary tree B whose nodes are labeled from 1 to n in symmetric order. A *block* of B is an interval $[i, j] \subseteq [1, n]$ of nodes in B. Any block $[i, j]$ of B induces a binary tree $B_{[i,j]}$, called the *block tree* of block $[i, j]$, which comprises exactly the nodes i to j. The root of $B_{[i,j]}$ is the lowest common ancestor of nodes i and j in B. The left child of a node x in $B_{[i,j]}$ is the highest node in the left subtree of x in B which lies in block $[i, j]$. The right child of a node in $B_{[i,j]}$ is defined analogously. Notice that, for any fixed subtree of B, the highest node of the subtree which lies in block $[i, j]$ is unique whenever it exists: if two such equally highest nodes existed, then their lowest common ancestor in the subtree would be higher than the two nodes and also lie in block $[i, j]$, which is a contradiction. How does a rotation in B affect a block tree $B_{[i,j]}$? If both the nodes involved in the rotation are in $B_{[i,j]}$, then the rotation translates into a rotation in $B_{[i,j]}$ involving the same pair of nodes. Otherwise, $B_{[i,j]}$ is unaffected.

[3]The *left* and *right paths* of a binary tree are, respectively, the paths from the root to the smallest node and the largest node in the tree.

It is easy to see that $U_1(n) = C_1(n) = \binom{n}{2} \leq n\hat{a}_0(n)$. Let us prove that $U_2(n) = O(n\hat{a}_1(n))$. Consider any RT-sequence performed on a binary tree B. We divide B into a left block $[1, \lfloor n/2 \rfloor]$ and a right block $[\lfloor n/2 \rfloor + 1, n]$. Every 2-turn either involves nodes from only one block (*intrablock*) or involves nodes from both blocks (*interblock*). An intrablock 2-turn effects a 2-turn in the corresponding block tree and gets counted in the RT-sequence for the block tree. Every interblock 2-turn either adds a node to the right path of the left block tree or deletes a node from the left path of the right block tree (See Fig. 4). For any binary tree, a right rotation on the tree does not increase the size of the left path of the tree and does not decrease the size of the right path of the tree. It follows that the number of interblock 2-turns is at most $n - 2$. This leads to the following recurrence for $U_2(n)$:

$$U_2(n) \leq \begin{cases} U_2(\lfloor n/2 \rfloor) + U_2(\lceil n/2 \rceil) + n - 2 & \text{if } n \geq 3 \\ 0 & \text{if } 1 \leq n \leq 2 \end{cases}$$

Solving the recurrence yields the desired bound for $U_2(n)$:

$$U_2(n) \leq n\lceil \log n \rceil - 2^{\lceil \log n \rceil} - n + 2 \leq n\hat{a}_1(n).$$

With a slight modification the same proof works for 2-cascades also. An interblock 2-cascade either decreases the size of the left path of the right block tree or increases the number of left block nodes whose left depth relative to the block is at most 1 (See Fig. 5). As a right rotation does not increase the left depth of any node, it follows that the number of interblock 2-cascades is bounded by $n - 3$. The bound $C_2(n) \leq n\hat{a}_1(n)$ follows.

Next, we extend the previous argument and obtain the bounds for $U_k(n)$ and $C_k(n)$ when $k \geq 3$.

Lemma 1 *When $m \geq n$, $\lfloor m/n \rfloor U_k(n) \leq U_k(m)$, for all k.*

Proof. Given an RT-sequence S for an n-node binary tree B that comprises $U_k(n)$ k-turns, construct a new tree of size $\lfloor m/n \rfloor n \leq m$ by starting with a copy of B and successively inserting a new copy of B as the right subtree of the largest node in the current tree $\lfloor m/n \rfloor - 1$ times. Since S can be performed on each of the copies of B one after another, there exists an RT-sequence with $\lfloor m/n \rfloor U_k(n)$ k-turns for a tree of size m. The lemma follows immediately. ■

Define a hierarchy of functions $\{K_i | i \geq 1\}$ as follows:

$$\begin{aligned} K_1(j) &= 8j \quad \text{for all } j \geq 1 \\ K_2(j) &= 2^{4j} \quad \text{for all } j \geq 1 \\ K_i(j) &= \begin{cases} iK_{i-2}(\lfloor i/2 \rfloor) & \text{if } i \geq 3 \text{ and } j = 1 \\ K_i(j-1)K_{i-2}(K_i(j-1)/4)/2 & \text{if } i \geq 3 \text{ and } j \geq 2 \end{cases} \end{aligned}$$

The following lemma establishes that the function K_i grows faster than the Ackerman function $A_{\lfloor i/2 \rfloor}$.

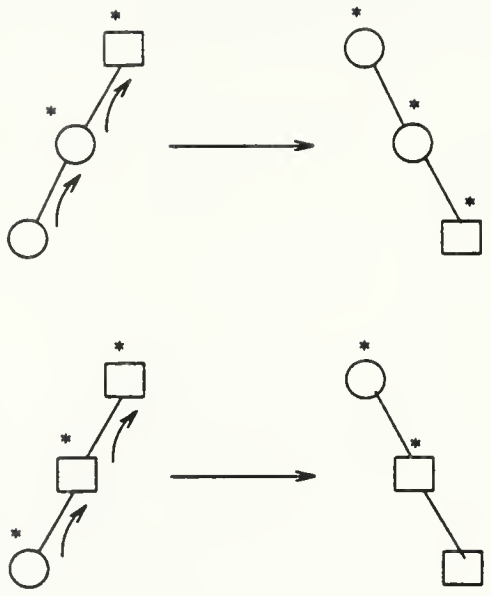


Figure 4: Two types of interblock right 2-turns. Circles denote left block nodes and squares denote right block nodes. Nodes that lie on the left/right path of their block are starred.

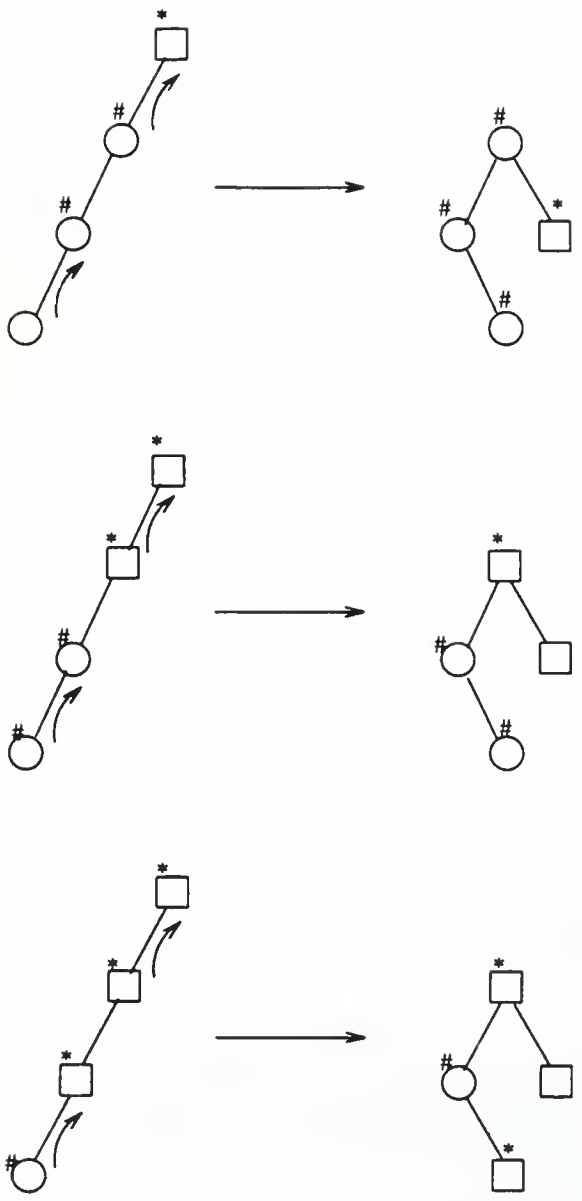


Figure 5: Three types of interblock 2-cascades. Left block nodes whose left depths are at most 1 are sharped and right block nodes lying on the left path of their block are starred.

Lemma 2 *1.* $A_1^{(2)}(j) \leq K_3(j)$ *for all* $j \geq 1$.

2. $A_{\lfloor i/2 \rfloor}(j) \leq K_i(j)$ *for all* $i \neq 3$ *and* $j \geq 1$.

Proof. Proposition 1 follows from the fact $K_3(j) = 24^{2^{j-1}}$.

It suffices to prove Proposition 2 when i is even since the proposition holds for $i = 1$ and $K_i(j) \leq K_{i+1}(j)$ for all $i \geq 1$ and $j \geq 1$. We will prove the inequality $4A_i(j) \leq K_{2i}(j)$ for all $i \geq 1$ and $j \geq 1$. The proof uses double induction on i and j and involves three cases:

Case 1. $i = 1$: $K_2(j) = 2^{4j} \geq 4A_1(j)$.

Case 2. $i \geq 2$ and $j = 1$:

$$\begin{aligned} K_{2i}(1) &= 2iK_{2i-2}(i) \\ &\geq 8iA_{i-1}(i) \quad \text{(By the Induction Hypothesis)} \\ &\geq 4A_i(1). \end{aligned}$$

Case 3. $i \geq 2$ and $j \geq 2$:

$$\begin{aligned} K_{2i}(j) &= K_{2i}(j-1)K_{2i-2}(K_{2i}(j-1)/4)/2 \\ &\geq 8A_i(j-1)A_{i-1}(A_i(j-1)) \quad \text{(By the Induction Hypothesis)} \\ &\geq 4A_i(j). \blacksquare \end{aligned}$$

When n is of the form $K_k(j)$, the following lemma derives the bound for $U_k(n)$.

Lemma 3 $U_k(K_k(j)) \leq 4jK_k(j)$, *for all* $k \geq 1$ *and* $j \geq 1$.

Proof. We use double induction on k and j.

Case 1. $1 \leq k \leq 2$: The lemma follows from the bounds $U_1(n) \leq n\hat{\alpha}_0(n)$ and $U_2(n) \leq n\hat{\alpha}_1(n)$.

Case 2. $k \geq 3$ and $j = 1$: We need to show that $U_k(K_k(1)) \leq 4K_k(1)$. Consider a binary tree B having $K_k(1)$ nodes on which an RT-sequence is performed. Divide B into a sequence of $K_{k-2}(\lfloor k/2 \rfloor)/2$ blocks of size $2k$ each. Each k-turn is of one of the following types:

Type A. All nodes involved in the k-turn are from the same block: Since a block has only $2k$ nodes, there is at most one such k-turn per block.

Type B. Some two nodes of the k-turn are from the same block: Let C denote the block tree of this block. The k-turn causes either an increase in the size of the right path of C, or a decrease in the size of the left path of C, or both. Hence the number of type B k-turns is at most $2K_k(1)$.

Type C. Each node of the k-turn is from a different block: To handle this case, we label the root of each block *global*. The global nodes in B induce a binary tree G, called the *global tree*. The root of G is identical to the root of B. The left child of a node x in G is the highest global node in the left subtree of x in B. The right child of a node is defined similarly. It is easy to see that the left and right children of any node in G are unique. The effect of a rotation in B on G is analogous to the effect of such a rotation on a block tree

of B: A rotation in B translates into a rotation in G if both the nodes of the rotation are global; otherwise, G is unaffected. (If a rotation changes the root of a block then the global role passes from the old root to the new root but this does not affect the global tree.)

Suppose that the k-turn turns the left subpath $x_1 - x_2 - \cdots - x_{k+1}$ of B into a right subpath. Since all x_is are from different blocks, the nodes $x_2, x_3, \ldots, x_k$ are all global. Therefore, the k-turn results in a $(k-2)$-turn in G (if x_1 or x_{k+1} is also global, then some right rotations are also performed in G.) The number of $(k-2)$-turns in G is at most

$$\begin{aligned} U_{k-2}(K_{k-2}(\lfloor k/2 \rfloor)/2) &\leq U_{k-2}(K_{k-2}(\lfloor k/2 \rfloor))/2 \quad \text{(By Lemma 1)} \\ &\leq 2\lfloor k/2 \rfloor K_{k-2}(\lfloor k/2 \rfloor) \quad \text{(By the Induction Hypothesis)} \\ &< K_k(1). \end{aligned}$$

This gives an upper bound of $K_k(1)$ for the number of type C k-turns in B.

Summing together the above bounds for the three types of k-turns, we obtain a bound of

$$K_{k-2}(\lfloor k/2 \rfloor)/2 + 2K_k(1) + K_k(1) \leq 4K_k(1)$$

for the total number of k-turns in the RT-sequence. This completes Case 2.

Case 3. $k \geq 3$ and $j \geq 2$: Here we divide the binary tree on which the RT-sequence is executed into $K_k(j)/K_k(j-1)$ blocks of size $K_k(j-1)$ each. We split the k-turns into the three types defined in Case 2 and obtain the following tally for each type of turn:

Number of type A k-turns

$$\begin{aligned} &\leq \frac{K_k(j)}{K_k(j-1)} \cdot 4(j-1)K_k(j-1) \quad \text{(By the Induction Hypothesis)} \\ &\leq 4(j-1)K_k(j). \end{aligned}$$

Number of type B k-turns $\leq 2K_k(j)$.

Number of type C k-turns

$$\begin{aligned} &\leq U_{k-2}(K_{k-2}(K_k(j-1)/4)/2) \\ &\leq U_{k-2}(K_{k-2}(K_k(j-1)/4))/2 \quad \text{(By Lemma 1)} \\ &\leq K_k(j-1)K_{k-2}(K_k(j-1)/4)/2 \quad \text{(By the Induction Hypothesis)} \\ &= K_k(j). \end{aligned}$$

Hence the total number of k-turns in the sequence is at most

$$4(j-1)K_k(j) + 2K_k(j) + K_k(j) < 4jK_k(j).$$

This finishes Case 3. ■

Using the three lemmas we obtain the bound for $U_k(n)$ for any k and n:

Theorem 1

$$U_k(n) \leq \begin{cases} 8n\hat{\alpha}_{\lfloor k/2 \rfloor}(n) & \text{if } k \neq 3 \\ 8n \log\log n & \text{if } k = 3 \end{cases}$$

Proof. Choose $j = \min\{l \geq 1 | K_k(l) \geq n\}$. We have

$$\begin{aligned} U_k(n) &\leq \frac{U_k(K_k(j))}{\lfloor K_k(j)/n \rfloor} \quad \text{(By Lemma 1)} \\ &\leq \frac{4jK_k(j)}{\lfloor K_k(j)/n \rfloor} \quad \text{(By Lemma 3)} \\ &\leq 8jn. \end{aligned}$$

Consider the case $k \neq 3$. If $j = 1$, $U_k(n) \leq 8n \leq 8n\hat{\alpha}_{\lfloor k/2 \rfloor}(n)$. Otherwise, we proceed as follows:

$$\begin{aligned} A_{\lfloor k/2 \rfloor}(j-1) &\leq K_k(j-1) \quad \text{(By Lemma 2)} \\ &< n. \end{aligned}$$

Since this implies that $j \leq \hat{\alpha}_{\lfloor k/2 \rfloor}(n)$, we have $U_k(n) \leq 8jn \leq 8n\hat{\alpha}_{\lfloor k/2 \rfloor}(n)$. This completes the case $k \neq 3$. The case $k = 3$ is handled similarly. ■

Theorem 1 is extended to cascades by the following theorem:

Theorem 2

$$C_k(n) \leq \begin{cases} 8n\hat{\alpha}_{\lfloor k/2 \rfloor}(n) & \text{if } k \neq 3 \\ 8n \log\log n & \text{if } k = 3 \end{cases}$$

Proof. Note that Lemma 1 still holds when U_k is replaced by C_k. Therefore the theorem can be proved in the same fashion as Theorem 1 once we prove Lemma 3 for $C_k(n)$. That is, we need to show that $C_k(K_k(j)) \leq 4jK_k(j)$, for all $k \geq 1$ and $j \geq 1$. Referring to the proof of Lemma 3, only the handling of cases 2 and 3 has to be modified. Consider Case 2. As before, the blocks have size $2k$ each. The k-cascades are categorized as follows:

Type A. All nodes involved in the cascade are from the same block: There is at most one type A cascade per block.

Type B. One of the cascade rotations involves a pair of nodes from the same block: If the cascade rotates an edge that lies on the left path of some block, then the length of the left path of the block decreases by at least 1. Alternately, if the lowest three nodes involved in the cascade are from the same block, then the number of nodes in that block whose left depth is at most 1 increases. We conclude that the number of type B cascades falling under the above categories is at most $2K_k(1) - K_{k-2}(\lfloor k/2 \rfloor)$. In every remaining type B k-cascade, only the lowest cascade rotation is intrablock and the lowest three nodes do not belong to the same block. Each such cascade behaves like a type C cascade in that it causes a $(k-2)$-cascade on the global tree (defined below) which accounts for it.

Type C. The nodes participating in any cascade rotation are from distinct blocks: In this case for each block, in addition to the root of the block, we also label the left child of the root in the block global, if it exists (if the root has no left child, then the right child of the root is global.) Right rotations are propagated from the original tree to the global tree as described in Lemma 3 except in the following situation: When the edge joining the root

and its left child, say l, in a block is rotated, the left child of l, say ll, now becomes global, and if ll is not adjacent to l in B, this results in a series of right rotations in the global tree (See Fig. 6). Under this definition of global tree, the $(k-2)$ interior rotations performed by any type C k-cascade are all global. Hence, each type C k-cascade translates into a right $(k-2)$-cascade and a set of right rotations in the global tree.
Therefore the total number of k-cascades in the sequence is at most

$$K_{k-2}(\lfloor k/2 \rfloor)/2 + 2K_k(1) - K_{k-2}(\lfloor k/2 \rfloor) + C_{k-2}(K_{k-2}(\lfloor k/2 \rfloor)) < 4K_k(1).$$

This completes Case 2 in the proof of Lemma 3 for cascades. Case 3 is handled similarly. ■

2.2 Lower bounds

As in the upper bound construction, we need a new Ackerman-like hierarchy of functions to prove the lower bounds for $U_k(n)$. Define:

$$\begin{aligned}
B_1(j) &= j \quad \text{for all } j \geq 1 \\
B_2(j) &= 2^j - 1 \quad \text{for all } j \geq 1 \\
B_i(j) &= \begin{cases} 1 & \text{if } i \geq 3 \text{ and } j = 1 \\ ((i+1)jB_i(j-1)+1)B_{i-2}((i+1)jB_i(j-1)) & \text{if } i \geq 3 \text{ and } j \geq 2 \end{cases}
\end{aligned}$$

We can show that the function B_i grows roughly at the same rate as the Ackerman function $A_{\lfloor i/2 \rfloor}$.

Lemma 4 *1. $B_3(j) \leq A_1^{(2)}(2j)$, for all $j \geq 1$.*
2. $B_i(j) \leq A_{\lfloor i/2 \rfloor}(3j)$ for all $i \neq 3$ and $j \geq 1$.

Proof. Both propositions are proved using induction. ■
The following lemma gives the lower bound for $U_k(n)$ when n is of the form $B_k(j)$:

Lemma 5 *$U_k(B_k(j)) \geq (1/2)(j-3)B_k(j)$ for all $k \geq 1$ and $j \geq 1$.*

Proof. We begin by introducing some definitions needed in the proof. For any positive integer k, a *k-turn-deletion sequence* (*k-UD-sequence* henceforth) is defined to be an intermixed sequence of right rotations, right k-turns and deletions of the minimum node performed on a binary tree. A *left path tree* is a binary tree comprising only left edges. A *right path tree* is defined similarly. Consider a right k-turn that is performed on some subpath $x_0 - x_1 - \cdots - x_k$ of a binary tree, such that x_0 is the lowest node on the subpath. x_0 is called the *base* of the turn. If x_k is the left child of a node y (say) in the tree, then the turn is called an *apex turn* and y is the *apex* of the turn. Otherwise, the turn is said to be *apexless.*

For any pair of positive integers k and j, we show that there exists a k-UD-sequence for the left path tree of $B_k(j)$ nodes satisfying the following properties:

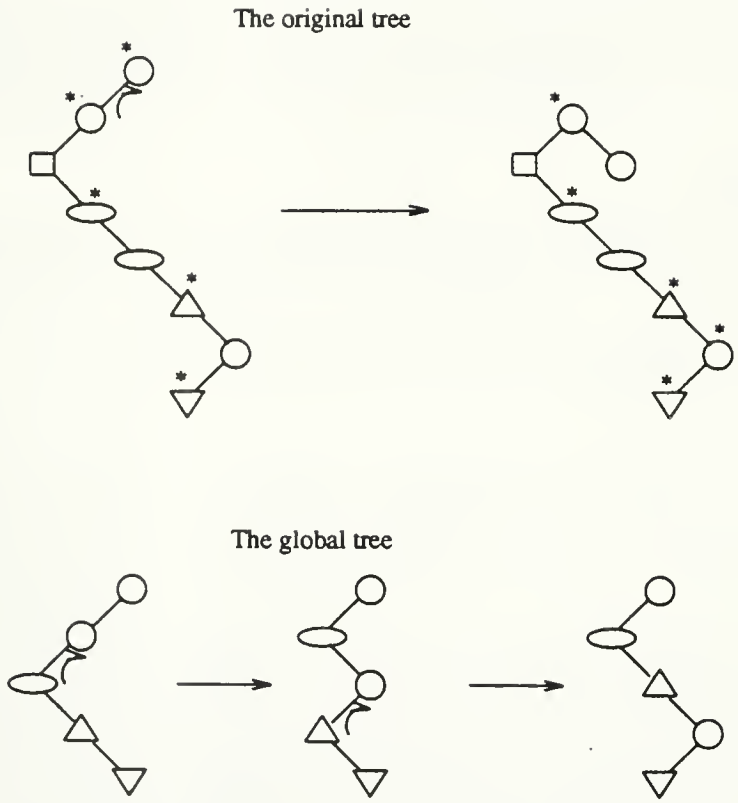


Figure 6: The effect of a rotation involving the root and its left child in a block on the global tree. The nodes of the block are denoted by circles and nodes from other blocks are denoted by other symbols. Starred nodes in the original tree are the global nodes.

1. The sequence deletes all the nodes from the tree.

2. A right k-turn always involves the minimum node of the tree.

3. A deletion always deletes the root of the tree.

4. The sequence comprises at least $(1/2)(j-3)B_k(j)$ apex k-turns. Further, if $k \geq 3$, there are no apexless k-turns in the sequence.

5. For any node x, the number of apex k-turns with base x is at most j.

6. For any node x, the number of apex k-turns with apex x is at most j.

The removal of all the deletions from a k-UD-sequence satisfying properties 1–6 would result in an RT-sequence consisting of at least $(1/2)(j-3)B_k(j)$ k-turns that converts a left path tree of $B_k(j)$ nodes into a right path tree. Therefore it suffices to prove the existence of such k-UD-sequences.

The construction of k-UD-sequences satisfying the above properties is achieved using double induction on j and k and involves the following cases:

Case 1. $k = 1$: The 1-UD-sequence repeats the following operation on the tree until the tree is empty: If the minimum node is the root, delete it; otherwise rotate the edge joining the minimum node and its parent. It is straightforward to check that the sequence obeys all the six properties.

Case 2. $k = 2$: Divide the left path tree into a lower left subpath comprising $2^{j-1} - 1$ nodes, a middle node, and an upper left subpath comprising $2^{j-1} - 1$ nodes. Recursively perform a 2-UD-sequence on the lower subpath. Modify each deletion in this sequence as follows: Perform a 2-turn on the subpath defined by the deleted node, say x, its parent (the middle node), and its grand parent; make x the root of the tree by successively rotating the edge joining it and its parent; delete x from the tree (this also deletes x from the lower subpath.) Next, delete the middle node which is currently the root of the tree. Finally, recursively perform a 2-UD-sequence on the upper subpath. (Fig. 7 illustrates a modification of the construction that derives the lower bound for $U_2(n)$ directly.)

This sequence performs $(j-2)2^{j-1} + 1$ 2-turns of which exactly $j-1$ are apexless. Therefore the number of apex 2-turns is at least $(j-3)2^{j-1} \geq (1/2)(j-3)B_2(j)$. This proves property 4. The remaining properties can be verified easily.

Case 3. $k \geq 3$ and $j = 1$: Since the tree is singleton the sequence simply deletes the root of the tree.

Case 4. $k \geq 3$ and $j \geq 2$: Let $s = (k+1)jB_k(j-1)$. The following lemma will be needed in the construction:

Lemma 6 *There exists a k-UD-sequence for a left path tree of size $s+1$ satisfying properties 1 and 2 and the following properties:*

$\bar{3}$. A deletion that is not the last operation in the sequence always deletes the left child of the root.

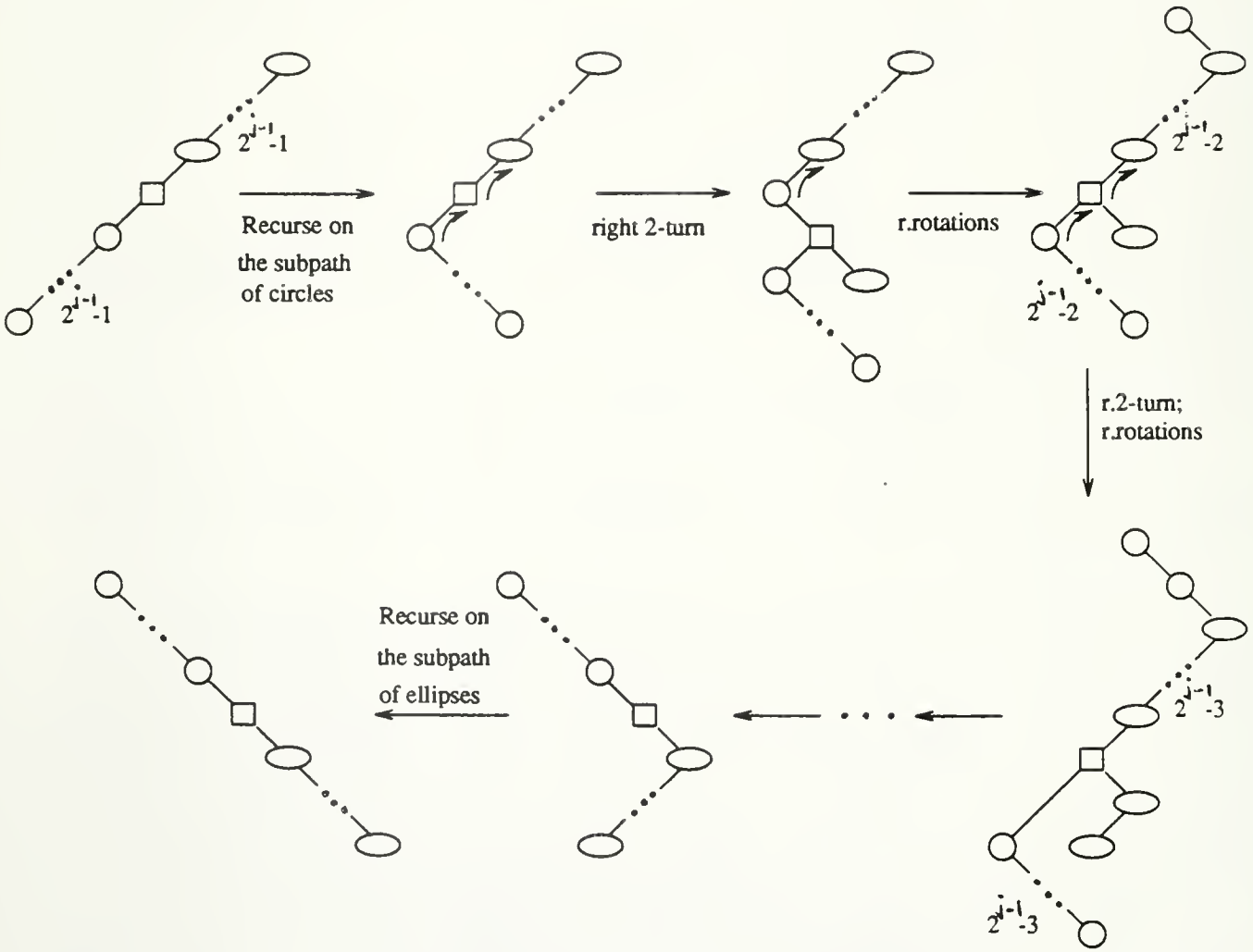


Figure 7: Lower bound construction for $U_2(n)$. The construction recursively transforms a left path tree of size 2^i-1 into a right path tree.

$\bar{4}$. *The sequence comprises at least* $(1/2)(j-4)s+j$ *right* k*-turns all of which are apex turns.*

$\bar{5}$. *For any node* x*, the number of apex* k*-turns with base* x *is at most* $j-1$.

$\bar{6}$. *For any node* x*, the number of apex* k*-turns with apex* x *is at most* $j-1$.

$\bar{7}$. *The root of the tree is always the maximum node.*

Proof. Divide the nodes of the tree excluding the root into a sequence of $(k+1)j$ blocks of size $B_k(j-1)$ each. Perform a k-UD-sequence obeying properties 1–6 on the lowest block. (The inductive hypothesis implies the existence of such a sequence.) Denote this sequence by S. Each deletion in S except the last is modified by rotating the deleted node up the tree until it is adjacent to the root and then deleting it. The deletion of the last node in the block, say x_1, is implemented differently. x_1 is rotated up the tree until it is in contact with the root of the next higher block, say x_2. x_2 is rotated in a similar fashion in order to make it adjacent to the root of the next higher block, say x_3. In this manner we create a left subpath $x_1 - x_2 - \cdots - x_{k+1}$ containing the roots of the lowest $k+1$ blocks. Next, a right k-turn is performed on this subpath and then x_1 is rotated up the tree and deleted. Following this, S is executed on the blocks of nodes $x_2, x_3 \ldots, x_{k+1}$ in succession. Each deletion is modified by first making the deleted node adjacent to the root and then deleting it. At the conclusion of this sequence of operations, all nodes in the lowest $k+1$ blocks of the tree have been deleted and at least $(1/2)(j-4)(k+1)B_k(j-1)+1$ apex k-turns have been performed. The above sequence of operations is repeated on each group of $k+1$ consecutive blocks, choosing the lowest group of blocks currently in the tree each time. The final operation of the sequence deletes the tree root.

It is obvious that the k-UD-sequence constructed above satisfies properties 1,2,$\bar{3}$ and $\bar{7}$. Since there are j groups of $k+1$ blocks each, the total number of apex k-turns executed by the sequence is at least $(1/2)(j-4)s+j$. Further, by property 4 of S, the sequence performs only apex k-turns. This proves property $\bar{4}$. Properties $\bar{5}$ and $\bar{6}$ are easy to show using properties 5 and 6 of sequence S. ■

Next we construct a k-UD-sequence for the left path tree of size $B_k(j)$ satisfying the six properties. The tree is partitioned into $B_{k-2}(s)$ blocks of size $s+1$ each. The root of each block is labeled global. The global nodes form a global tree as described in the proof of Lemma 3. By the induction hypothesis, there exists a $(k-2)$-UD-sequence, say $\bar{S}$, for the global tree satisfying properties 1-6. We construct the k-UD-sequence, denoted S, for the original tree by mapping each global tree operation in $\bar{S}$ onto a sequence of original tree operations, preserving the correspondence between the two trees. The following invariants define the relationships between the two trees:

A. Let B denote the block containing the minimum node in the tree and let x denote the root of B. Suppose that d nodes have been deleted from B so far. Then,

   i. The number of apex $(k-2)$-turns performed so far on the global tree that had x as their base is exactly d.

   ii. Denote by $\hat{S}$ the k-UD-sequence constructed by Lemma 6 that deletes all nodes from the left path tree of size $s+1$. Let T denote the tree that results when the

prefix of $\hat{S}$ up to the d^{th} deletion is executed on the left path tree. The block tree of B equals T.

B. Consider any block B that does not contain the minimum node of the tree. Let x denote the root of B. The block tree of B is a left path tree which is divided into the root and two subpaths. The nodes in the lower subpath, called *black nodes,* are the nodes in B that have participated in a k-turn. The nodes in the upper subpath are called *white nodes.*

   i. If b denotes the number of black nodes currently in B, then exactly b of the apex $(k-2)$-turns performed so far on the global tree had x as their apex.

C. If x is a global node with a right child y in the global tree, y is also the right child of x in the original tree.

D. If x is a global node with a left child y in the global tree, there is a left subpath $x = x_0 - x_1 - \ldots - x_{k+1} = y$ in the original tree such that $x_1, x_2, \ldots, x_k$ are the set of white nodes in the block of x.

Each global tree operation in $\bar{S}$ is simulated as follows:

Right rotation: Suppose that a global tree edge $[x, y]$, such that y is a left child of x, is rotated. In the original tree we repeatedly rotate the edge connecting y and its parent until x becomes the right child of y. Only invariants C and D are affected by the rotations. It is not hard to see that both these invariants are true after the last rotation.

Deletion: Suppose that a global node x is deleted. Since x is the root of the global tree, it is also the root of the original tree. Let d denote the number of nodes deleted so far from the block of x. We perform $\hat{S}$ (the sequence constructed by Lemma 6) on the block tree of x starting immediately after the d^{th} deletion. Each deletion is modified so that the deleted node is first made the root of the tree and then deleted. Invariant A.ii ensures that this is valid and that this will result in the deletion of all the nodes in the block of x from the tree. Therefore this sequence of operations reestablishes the correspondence between the global tree and the original tree.

Apexless $(k-2)$-turn: Break up the turn into a sequence of $k-2$ global rotations and simulate each global rotation as specified above.

Apex $(k-2)$-turn: Suppose that a global tree subpath $x_1 - x_2 - \cdots - x_{k-1}$ is turned and that x_1 is the base of the turn. Let x_0 denote the smallest node in the block of x_1 and let x_k denote the parent of x_{k-1} in the original tree. We create the subpath $x_0 - x_1 - \ldots - x_k$ in the original tree and perform a k-turn on this subpath. This is implemented as follows:

1. Let d denote the number of nodes deleted so far from the block of x_1. Execute the segment of sequence $\hat{S}$ between the d^{th} and the $(d+1)^{\text{st}}$ deletions (excluding the deletions) on the block tree of x_1. By Lemma 6, property $\bar{3}$, this makes node x_0 the left child of x_1.
2. Rotate x_1 up the tree until its parent is x_2. Continuing in this fashion, make the nodes $x_2, x_3, \ldots, x_{k-1}$ adjacent.
3. Perform a k-turn on the subpath $x_0 - x_1 - \cdots - x_k$.
4. Rotate x_0 up the tree making it the root and delete it.
5. Since x_k has become black due to the k-turn, the edge joining x_k and its left child is repeatedly rotated until the left child of x_k is not a global node.

Invariant B.i and property 6 of $\bar{S}$ guarantee that x_k is white at the beginning of this sequence of operations. Similarly, invariant A.i and property 5 of $\bar{S}$ guarantee that x_0 is well defined. Observe that all invariants are true at the end of the simulation.

The sequence of operations performed on the original tree during the simulation of $\bar{S}$ constitutes sequence S.

S deletes all the nodes in the original tree since $\bar{S}$ deletes all the nodes in the global tree. This proves that S satisfies property 1.

Properties 2 and 3 of S are apparent from the simulation procedure.

By Lemma 6, at least $(1/2)(j-4)s + j$ apex k-turns (*local turns*) are performed during the execution of $\hat{S}$ on any particular block. Hence the total number of local turns summed over all blocks is at least $(1/2)(j-4)sB_{k-2}(s) + jB_{k-2}(s)$. The number of turns involving global nodes (*global turns*) equals the number of $(k-2)$-turns in $\bar{S}$ which, by the Induction Hypothesis, is at least $(1/2)(s-3)B_{k-2}(s)$. Therefore the total number of k-turns in S is at least

$$\begin{aligned}
&(1/2)(j-4)sB_{k-2}(s) + jB_{k-2}(s) + (1/2)(s-3)B_{k-2}(s) \\
&= ((1/2)(j-3)s + j - (3/2))B_{k-2}(s) \\
&> (1/2)(j-3)B_k(j).
\end{aligned}$$

Evidently, every k-turn in S has an apex. This proves property 4.

For any node x, there is at most one global turn with x as the base since x is deleted from the tree immediately after the turn. By Lemma 6, there are at most $j-1$ local turns with x as the base. We conclude property 5. Property 6 is proved analogously. ■

The lower bound for $U_k(n)$ when n is arbitrary is stated in the next theorem.

Theorem 3

$$U_k(n) \geq \begin{cases} (1/12)n\hat{\alpha}_{\lfloor k/2 \rfloor}(n) - O(n) & \text{if } k \neq 3 \\ (1/8)n \log\log n - O(n) & \text{if } k = 3 \end{cases}$$

Proof. Let $j = \max\{l \geq 1 | B_k(l) \leq n\}$. Then

$$\begin{aligned} U_k(n) &\geq \lfloor n/B_k(l) \rfloor U_k(B_k(l)) \quad \text{(By Lemma 1)} \\ &\geq \frac{nU_k(B_k(l))}{2B_k(l)} \\ &\geq (1/4)n(l-3) \quad \text{(By Lemma 5).} \end{aligned}$$

Consider the case $k \neq 3$. By Lemma 4, $A_{\lfloor k/2 \rfloor}(3l+3) \geq B_k(l+1) \geq n$. Therefore $l \geq \hat{\alpha}_{\lfloor k/2 \rfloor}(n)/3 - 1$. The bound for $U_k(n)$ follows. The case $k = 3$ is handled similarly. ■

The lower bound for $C_k(n)$ is given by:

Theorem 4

$$C_k(n) \geq \begin{cases} (1/12)n\hat{\alpha}_{\lfloor k/2 \rfloor}(n) - O(n) & \text{if } k \neq 3 \\ (1/8)n \log\log n - O(n) & \text{if } k = 3 \end{cases}$$

Proof. We modify the lower bound proof for $U_k(n)$ given above. Define:

$$\begin{aligned} B'_1(j) &= j \quad \text{for all } j \geq 1 \\ B'_2(j) &= 3.2^j - 2 \quad \text{for all } j \geq 1 \\ B'_i(j) &= \begin{cases} 1 & \text{if } i \geq 3 \text{ and } j = 1 \\ (4ijB'_i(j-1)+3)B'_{i-2}(4ijB'_i(j-1)) & \text{if } i \geq 3 \text{ and } j \geq 2 \end{cases} \end{aligned}$$

It is easy check that Lemma 4 holds for the new hierarchy $\{B'_i\}$. We prove the analogue of Lemma 5 for $C_k(n)$, which states that $C_k(B'_k(j)) \geq (1/2)(j-3)B'_k(j)$ for all $k \geq 1$ and $j \geq 1$. We will use the notions of base and apex with regard to cascades just like they were used in the context of turns. We construct a sequence of right rotations, right k-cascades and deletions of the minimum node (a *k-CD-sequence* hereafter) that converts a left path tree of size $B'_k(j)$ into a right path tree and satisfies the counterparts of properties 1–6 for cascades.

Case 1. $k = 1$ or $j = 1$: Easy.

Case 2. $k = 2$: Divide the left path tree into a lower subpath and an upper subpath, each having $3.2^{j-1} - 2$ nodes, and two middle nodes. The 2-CD-sequence is constructed by recursing on the lower and upper subpaths in turn and performing a 2-cascade involving the deleted node and the middle nodes for each deletion in the first recursive step. It is straightforward to check that the sequence comprises $(3j-4)2^{j-1} - j + 2 \geq (1/2)(3.2^j - 2)(j-3)$ apex 2-cascades and satisfies all the properties.

Case 3. $k \geq 3$ and $j \geq 2$: Let $s = 4kjB'_k(j-1)$. A *p,q-zigzag tree* is a tree that is constructed from a p-node left path tree by inserting a q-node left path tree as the right subtree of the minimum node. We extend Lemma 6 to cascades and construct a k-CD-sequence, say $\hat{S}$, for a 3, s-zigzag tree that comprises $(1/2)(j-4)s+2j$ apex k-cascades. Each deletion in the sequence, except for the last two deletions, deletes the leftmost grandchild of the root. The proof divides the tree into $2j$ groups of $2k$ blocks, each of size $B'_k(j-1)$, and recursively performs a k-CD-sequence on each block, choosing the blocks in bottom-to-top

order. A k-cascade is performed on the roots of the blocks within each group, yielding $2j$ extra cascades.

The tree is partitioned into $B'_{k-2}(s)$ blocks of size $s+3$ each. The global tree is constructed from the roots of the blocks and a $(k-2)$-CD-sequence, say $\bar{S}$, satisfying properties 1–6 is recursively performed on it. The simulation of $\bar{S}$ on the original tree maintains the invariants A (with expression $s+3$ replacing $s+1$), C and D and the following modification of invariant B:

$\bar{\text{B}}$. Consider any block B that does not contain the minimum node of the tree. Let x denote the root of B. Block B induces a connected subtree in the original tree. Further, if exactly b of the $(k-2)$-cascades performed so far on the global tree had x as their apex, then the block tree of B is a $(s-b+3), b$-zigzag tree.

The simulation of global tree operations other than apex $(k-2)$-cascades is as before. Consider an apex $(k-2)$-cascade in $\bar{S}$ involving a global tree subpath $x_1 - x_2 - \cdots - x_{2k-4}$, such that x_1 is the base of the cascade. Let y_1 and y_2, respectively, denote the minimum node in the block of x_1 and the left child of x_1. Let z_1 and z_2, respectively, denote the parent and the grandparent of x_{2k-4} in the original tree. The global tree cascade is simulated on the original tree by creating the subpath $y_1 - y_2 - x_1 - x_2 - \cdots - x_{2k-4} - z_1 - z_2$ in the original tree, performing a k-cascade on this subpath, and finally deleting y_1 from the tree. We verify that the resulting sequence, say S, satisfies property 4:

$$\begin{aligned} \#\ k\text{-cascades in } S &= \#\text{ local cascades} + \#\text{ global cascades} \\ &\geq ((1/2)(j-4)s + 2j)B'_{k-2}(s) + (1/2)(s-3)B'_{k-2}(s) \\ &> (1/2)(j-3)B'_k(j). \end{aligned}$$

The rest of the properties of S are easy to check. This finishes the proof of Lemma 5 for cascades. Theorem 4 follows. ■

3 Bounds for twists

3.1 Upper bound

Define $L_i(j) = \binom{i+j-1}{i}$ for all $i \geq 1$ and $j \geq 1$. The following lemma derives the upper bound for $T_k(n)$ when n is of the form $L_k(j)$.

Lemma 7 *$T_k(L_k(j)) \leq k\binom{k+j-1}{k+1}$ for all $k \geq 1$ and $j \geq 1$.*

Proof. We use double induction on k and j.

Case 1. $k = 1$ or $j = 1$: Straightforward.

Case 2. $k \geq 2$ and $j \geq 2$: The tree is partitioned into a left block of $L_{k-1}(j)$ nodes and a right block of $L_k(j-1)$ nodes. If the lowest $k-1$ rotations of a k-twist in the RT-sequence involve only left block nodes, then these rotations perform a right $(k-1)$-twist on

the left block. Hence, by the induction hypothesis, there are at most $(k-1)\binom{k+j-2}{k}$ such k-twists in the RT-sequence. Similarly, the number of k-twists that involve only right block nodes is at most $k\binom{k+j-2}{k+1}$. Consider a k-twist that does not belong to these two categories. The highest rotation of such a twist involves only right block nodes; also, the lowest node involved in the twist is a left block node. This implies that the highest node of the twist is a right block node that leaves the left path of its block as a result of the twist. Therefore the number of k-twists in the third category is at most the initial size of the left path of the right block $\leq L_k(j-1) = \binom{k+j-2}{k}$. It follows that the total number of right k-twists in the RT-sequence is bounded by

$$\begin{aligned}
&(k-1)\binom{k+j-2}{k} + k\binom{k+j-2}{k+1} + \binom{k+j-2}{k} \\
&= k\binom{k+j-2}{k} + k\binom{k+j-2}{k+1} \\
&= k\binom{k+j-1}{k+1}.
\end{aligned}$$

■

Theorem 5 *$T_k(n) \leq kn^{1+1/k}$ for all $k \geq 1$ and $n \geq 1$.*

Proof. A straightforward calculation using the analogue of Lemma 1 corresponding to $T_k(n)$ and the preceding lemma proves the theorem. ■

3.2 Lower bound

Lemma 8 *$T_k(L_k(j)) \geq \binom{k+j-1}{k+1}$ for all $k \geq 1$ and $j \geq 1$.*

Proof. For any pair of positive integers k and j, we show that there exists a sequence of right k-twists, right rotations and deletions of the minimum node (a *k-TD-sequence* hereafter) performed on a left path tree of $L_k(j)$ nodes that has the following properties:

1. The sequence deletes all nodes from the tree.
2. A deletion always deletes the root of the tree.
3. A k-twist is always performed on the left path of the tree.
4. The sequence has exactly $\binom{k+j-1}{k+1}$ k-twists.

The removal of the deletions from such a sequence would yield an RT-sequence with the desired number of k-twists. The construction of these TD-sequences proceeds by double induction on k and j.

Case 1. $k = 1$ or $j = 1$: Easy.

Case 2. $k \geq 2$ and $j \geq 2$: Divide the left path tree into a lower block of size $L_{k-1}(j)$ and an upper block of size $L_k(j-1)$. Recursively perform a $(k-1)$-TD-sequence, say $\hat{S}$, on the lower block. For each $(k-1)$-twist in $\hat{S}$, first rotate the edge joining the root of the lower block with its parent and then perform the $(k-1)$-twist on the block. This is equivalent to a k-twist on the left path of the entire tree. Each deletion is modified by first making the deleted node the root of the tree using right rotations and then deleting the node. By property 4 of $\hat{S}$, the number of $(k-1)$-twists in $\hat{S}$ is exactly $\binom{k+j-2}{k}$. The initial depth of the root of the lower block equals $L_k(j-1) = \binom{k+j-2}{k}$. Since each $(k-1)$-twist in $\hat{S}$ reduces the depth of the root of the lower block by 1 and no other operation in $\hat{S}$ affects the depth, it is always possible to rotate the root of the lower block just before the execution of any $(k-1)$-twist in $\hat{S}$. The construction is completed by recursively performing a k-TD-sequence, say $\bar{S}$, on the upper block.

It is apparent from the construction that this sequence satisfies properties 1–3. The total number of k-twists performed by the sequence equals

$$\begin{aligned}
&(\text{the number of } (k-1)\text{-twists in } \hat{S}) + (\text{the number of } k\text{-twists in } \bar{S}) \\
&= \binom{k+j-2}{k} + \binom{k+j-2}{k+1} \quad (\text{By the Induction Hypothesis}) \\
&= \binom{k+j-1}{k+1}.
\end{aligned}$$

This proves property 4. ■

Theorem 6 *$T_k(n) \geq n^{1+1/k}/2e - O(n)$ for all $k \geq 1$ and $n \geq 1$.*

Proof. Similar to the proofs of the previous theorems. ■

4 Inverse Ackerman bound for Deque conjecture

Define the *cost* of a right twist to be the number of rotations performed by the twist. Similarly, the *cost* of performing a deque operation on a binary tree using splaying is defined to be the number of rotations performed by the operation. Since a deque operation operation performs only constant extra work besides the work required for rotations, this cost underestimates the true cost of a sequence of m operations by only $O(m)$.

Lemma 9 *Consider an arbitrary RT-sequence executed on an n-node binary tree. The total cost of any m right cascades in the sequence is $O((m+n)\alpha(m+n,n))$.*

Proof. Let $l = 2\alpha(m+n,n)+2$. Split each of the m right cascades into a sequence of right l-cascades followed by a sequence of at most $l-1$ rotations. By Theorem 2, the total number of right l-cascades is at most $8n\hat{\alpha}_{\lfloor l/2 \rfloor}(n)$. This yields a bound of $(m+8n\hat{\alpha}_{\lfloor l/2 \rfloor}(n))l$ for the number of rotations performed by the m right cascades. We bound $\hat{\alpha}_{\lfloor l/2 \rfloor}(n)$ as follows:

$$\begin{aligned} A_{\alpha(m+n,n)+1}(\lfloor m/n \rfloor + 2) &= A_{\alpha(m+n,n)}(A_{\alpha(m+n,n)+1}(\lfloor m/n \rfloor + 1)) \\ &\geq A_1(A_{\alpha(m+n,n)}(\lfloor (m+n)/n \rfloor)) \\ &\geq n. \end{aligned}$$

Therefore $\hat{\alpha}_{\lfloor l/2 \rfloor}(n) = \hat{\alpha}_{\alpha(m+n,n)+1}(n) \leq \lfloor m/n \rfloor + 2$. The lemma follows. ■

Remark. Hart and Sharir [3] proved a result similar to Lemma 9 concerning sequences of certain path compression operations on rooted ordered trees. In fact, their result can be derived using the counterpart of Lemma 9 for turns by interpreting turns in a binary tree as path compressions on the rooted ordered tree representation of the binary tree. It is interesting to note that they also use ideas similar to blocks and global tree in their proof.

Define a *left path rotation* to be a right rotation performed on the left path of a binary tree. A *right path rotation* is defined analogously.

Lemma 10 *Consider an intermixed sequence of* POPs, PUSHs, *left path rotations and right path rotations performed on an arbitrary n-node binary tree. The total cost of* POP *operations is $O((m+n)\bar{\alpha}(m+n))$, where m denotes the number of* POPs *and* PUSHs *in the sequence.*

Proof. We simplify the sequence through a series of transformations without undercounting POP rotations.

Simplification 1 *The first operation of the sequence is a* POP.

Transformation. If there are no POPs in the sequence, the lemma is trivial. Hence suppose that the sequence contains POPs and that the first operation is not a POP. Delete the operations preceding the first POP from the sequence and modify the initial tree by executing the deleted prefix of the sequence on it. The transformation does not change the number of POP rotations or the quantity $(m+n)\bar{\alpha}(m+n)$ defining the upper bound. ■

Simplification 2 *The sequence does not contain* PUSH *operations.*

Transformation. We show how to get rid of the first PUSH from the sequence and repeated applications of this transformation will completely eliminate all the PUSHs from the sequence. Suppose that the first PUSH pushes a node, say y, into the tree and that the most recent POP preceding the PUSH pops some node, say x, from the tree. Modify the initial tree as follows: If the right child of x is empty insert y as the right child of x; Otherwise insert y as the left child of the symmetric order successor of x. All operations except PUSH(y) are performed as in the original sequence. y appears on the left path for

the first time when x is popped. PUSH(y) is implemented by making y the new root of the tree using right rotations. The rest of the sequence is identical to the original sequence. ■

Define a PARTIALPOP to be a sequence of arbitrarily many right 2-turns performed on the minimum node followed by deletion of the minimum node.

Simplification 3 *The sequence consists of only* PARTIALPOP*s and left path rotations and the lemma is true if the total cost of* PARTIALPOP *operations is* $O((m+n)\bar{\alpha}(m+n))$*, where* m *denotes the number of* PARTIALPOP*s in the sequence and* n *denotes the size of the initial tree.*

Transformation. Modify the initial tree by performing a left turn that spans the right path of the tree. The left path of the new tree is identical to the path between the smallest and the largest node in the old tree. Further, except for the nodes on the right path of the old tree, the subtree rooted at every other node is identical in two trees. If a node x on the right path of the old tree has a parent p and a left subtree T, then T is the right subtree of p in the new tree.

We simulate the sequence of old tree operations on the new tree, preserving the above-stated correspondence between the two trees. A POP on the old tree is equivalent to a PARTIALPOP on the new tree. However the PARTIALPOP does not execute the last splay step if it is a single rotation. Since the number of such rotations over the entire sequence is at most m, there is no problem due to undercounting of rotations. A left path rotation in the old tree maps onto a left path rotation of the corresponding edge in the new tree. An exception occurs when the root is involved in the rotation: In this case, the new tree remains as is. A right path rotation that rotates an edge $[x, p]$, such that p is the parent of x and p has a grandparent g, translates into rotation of the left path edge $[p, g]$ in the new tree. This completes the simulation. ■

Simplification 4 *The sequence comprises only right cascades and left path rotations. Further, the lemma is true if the total cost of any* m *cascades in the sequence is* $O((m+n)\bar{\alpha}(m+n))$.

Transformation. We simulate the sequence of operations on a new tree, maintaining the invariant that the original tree is a subtree of the new tree. A PARTIALPOP involving a subpath $x_0 - x_1 - \cdots - x_{2k}$ is simulated by first performing a right k-cascade spanning the subpath $x_1 - x_2 - \cdots - x_{2k}$ in the new tree and then moving x_0 up the tree using right rotations until it joins the right path. Since the cascade performs exactly half as many rotations as the PARTIALPOP, only a constant fraction of the PARTIALPOP rotations are ignored. A right rotation is simulated by rotating the corresponding edge in the new tree. ■

Lemma 10 follows from simplification 4 and Lemma 9. ■

The bound for the deque conjecture is derived by the following theorem:

Theorem 7 *The cost of performing an intermixed sequence of* m *deque operations on an arbitrary* n*-node binary tree using splaying is* $O((m+n)\bar{\alpha}(m+n))$.

Proof. Divide the sequence of operations into a series of epochs as follows: The first epoch comprises the first $\max\{\lfloor n/2 \rfloor, 1\}$ operations in the sequence. For all $i \geq 1$, if the tree contains k nodes at the end of epoch i, then epoch $i+1$ consists of the next $\max\{\lfloor k/2 \rfloor, 1\}$ operations in the sequence. The last epoch might consist of fewer operations than specified. It suffices to show that the cost of an epoch that starts with a k-node tree is $O(k\bar{\alpha}(k))$, since the sum of the sizes of the starting trees over all epochs is $O(m+n)$.

Consider an epoch whose initial tree, say T, has $k \geq 2$ nodes. Divide T into a left block of $\lfloor k/2 \rfloor$ nodes and a right block of $\lceil k/2 \rceil$ nodes. This partitioning ensures that neither block gets depleted before the epoch completes. The total cost of PUSHs and INJECTs is 0, since only rotations contribute to the operation cost. We show that the total cost of POPs is $O(k\bar{\alpha}(k))$. The same proof will apply to EJECTs.

A POP on T translates into a POP on the left block. The effect of a POP on the right block is a series of left path rotations. It is easy to see that the total number of rotations performed by the POP is at most

(the number of rotations performed by the POP in the left block) +
2(the number of left path rotations performed on the right block) + 2.

A PUSH operation on the tree propagates as a PUSH on the left block. An EJECT performs only right path rotations on the left block. An INJECT does not affect the left block. Hence by Lemma 10, the total number of rotations performed by all the POPs on the left block is $O(2\lfloor k/2 \rfloor \bar{\alpha}(2\lfloor k/2 \rfloor)) = O(k\bar{\alpha}(k))$. A left path rotation on the right block decreases the size of the left path of the block by 1. The initial size of this path is at most $\lceil k/2 \rceil$ and the size increases by at most 1 per deque operation. Therefore the total number of left path rotations performed on the right block due to POPs is at most k. This leads to an $O(k\bar{\alpha}(k))$ bound on the total cost of all the POPs performed during the epoch. ■

5 Upper bound for shallow twists

Lemma 11 *For any $d \geq 1$, the total number of d-shallow right twists in any RT-sequence is at most $4dn$.*

Proof. Consider any d-shallow right twist that rotates a sequence of edges, say $[x_1, y_1], [x_2, y_2], \ldots, [x_k, y_k]$, such that the left depths of the sequence of nodes $x_k, y_k, x_{k-1}, y_{k-1}, \ldots, x_1, y_1$ is nonincreasing. Let $ld(z)$ and $ld'(z)$, respectively, denote the left depth of any node z before and after the twist. For all $i \in [\lceil k/2 \rceil, k]$, we have

$$\frac{ld'(x_i)}{ld(x_i)} = 1 - \frac{i}{ld(x_i)} \leq 1 - \frac{i}{kd} \leq 1 - \frac{1}{2d}.$$

In order to pay unit cost for a twist, we charge each node x_i, such that $i \in [\lceil k/2 \rceil, k]$, $\min\{2d/ld(x_i), 1\}$ debits. Let us prove that the total charge is at least 1. If $ld(x_{\lceil k/2 \rceil}) \leq 2d$, then $x_{\lceil k/2 \rceil}$ is charged 1 debit. Otherwise, we have $1 \geq 2d/ld(x_i) \geq 2/k$ for all $i \geq \lceil k/2 \rceil$.

Since $\lfloor k/2 \rfloor + 1$ nodes are each charged at least $2/k$ debits, the net charge to all the nodes is at least 1.

Now, we bound the total charge to each node over the entire sequence. Call a node *deep* if its left depth is greater than $2d$ and *shallow* otherwise. Suppose that a node receives a sequence of charges $2d/L_k, 2d/L_{k-1}, \ldots, 2d/L_0$ while it is deep. Then

$$L_i > \frac{2d}{(1 - 1/2d)^i} \quad \text{for all } i \geq 0.$$

Therefore the total charge to a node while it remains deep is at most

$$(1 - 1/2d)^k + (1 - 1/2d)^{k-1} + \cdots + 1 \leq 2d.$$

A node receives at most $2d$ debits while it is shallow. This implies that any node is charged at most $4d$ debits, giving a bound of $4dn$ for the total number of d-shallow right twists. ■

Theorem 8 *$S^{(d)}(n) \leq 87dn$ for all $d \geq 1$ and $n \geq 1$.*

Proof. The proof uses induction on n.

Case 1. $n \leq 174d$: $S^{(d)}(n) \leq \binom{n}{2} \leq 87dn$.

Case 2. $n > 174d$: Divide the tree into a sequence of $\lceil n/K \rceil$ blocks such that each block except the first contains exactly $K = 29d$ nodes. The first block may contain fewer nodes. In each block except the first, the nodes with preorder numbers 1 to $4d$ within the block are global. The first block does not contain any global nodes. Notice that the global nodes of a block form a connected subtree within the block whose root coincides with the root of the block. Further, if the left path of the block contains more than $4d$ nodes then all global nodes lie on the left path of the block. Otherwise all nodes on the left path of the block are global. The global nodes in the tree form a global tree as in the previous upper bound constructions. The size of the global tree is at most $n/7.25$.

Next, we analyze the effect of a right rotation in the original tree on the global tree. An interblock right rotation translates into a rotation of the corresponding edge in the global tree if both nodes of the rotation are global. Otherwise the global tree remains as is. The analysis of an intrablock right rotation is more complicated and involves the following cases:

Local-local: The global tree is unaffected.

Local-global: Again, the global tree is not affected, but the global role is transferred from the global node to the local node.

Global-global: Let $[x, p]$ denote the rotated edge such that p is the parent of x. If the left subtree of x within the block contains only global nodes, then the rotation simply propagates to the global tree. Otherwise, p's global role is transferred to the node, say x', in p's block that had preorder number $4d + 1$ initially. Let p' denote the lowest ancestor of x' in the original tree which is global. The effect of the transfer of global role on the structure of the global tree is to contract edge $[x, p]$ and add a new edge

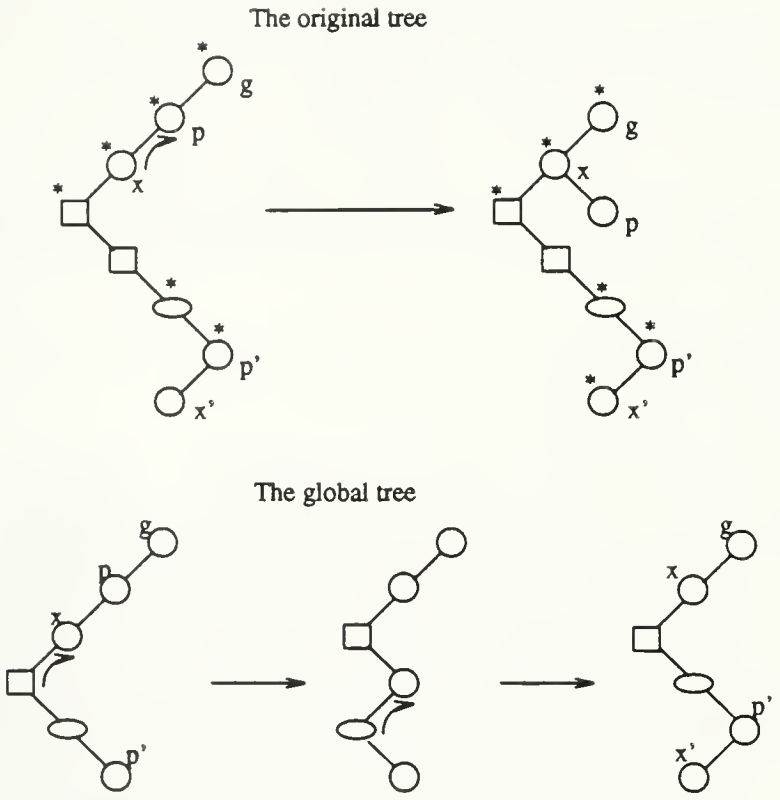


Figure 8: Global role transfer in an intrablock global-global rotation. The nodes of the block are denoted by circles. Global nodes are starred in the original tree.

$[x', p']$. We show that the same transformation is realizable through a series of right rotations in the global tree. These rotations are performed by traversing the path from p to p' in the global tree as follows (see Fig. 8):

> Start at edge $[p, x]$ and repeat the following operation until the last edge on the path is reached: If the next edge on the path is a left edge, move to the next edge; otherwise, rotate the current edge and move to the next edge after the rotation. Finally, if x' belongs to the right subtree of p' in the original tree, rotate the last global tree edge traversed.
>
> **Remark.** The operation performs all the rotations within the subtree of the global tree rooted at x. This is seen as follows. If $x = p'$, then no rotations are performed on the global tree. Otherwise, p' lies in the left subtree of x. Hence the successor of edge $[p, x]$ on the global tree path from p to p' is a left edge. This implies that the operation does not rotate edge $[p, x]$. Therefore all the rotations performed by the operation occur in the subtree of the global tree rooted at x.

The invariant maintained during this traversal is that the current edge is a left edge and the contraction of the current edge results in a tree identical to the tree obtained by contracting the edge $[x, p]$ in the initial tree. It follows that the above series of rotations simulates the rotation of edge $[x, p]$ correctly.

The above discussion leads to the following useful observation: The rotation of an edge $[x, p]$, such that p is the parent of x, either does not affect the global tree, or translates into a rotation of the edge $[x, p]$ in the global tree, or translates into a sequence of right rotations in the subtree of the global tree rooted at x. The following definition will be needed later. Call a rotation in the original tree *global* if it results in a rotation of the corresponding edge in the global tree and *local* otherwise.

Consider the effect of a right twist in the original tree on the global tree. The sequence of global tree rotations caused by the right twist comprises *l-rotations,* caused by local rotations in the twist, and *g-rotations,* caused by global rotations in the twist. We want to show that this sequence can be rearranged so that all the l-rotations precede all the g-rotations, preserving the effect of the sequence on the global tree. Consider any l-rotation and any g-rotation preceding the l-rotation. Evidently, the global rotation causing the g-rotation must occur before the local rotation causing the l-rotation. If x denotes the bottom node of the local rotation, then the g-rotation involves only proper ancestors of x in the global tree while the l-rotation affects only the subtree of the global tree rooted at x. This shows that the nodes involved in any l-rotation are distinct from the nodes involved in any previous g-rotation. Hence we may transform the sequence of global tree rotations by moving each l-rotation before all the g-rotations without altering the net effect of the sequence on the global tree. The suffix of the sequence consisting of all the g-rotations defines a *global twist* on the global tree. In summary, a right twist on the original tree may be simulated on the global tree by performing a sequence of right rotations followed by a global twist corresponding to the subsequence of global rotations in the twist.

Next, we estimate the number of rotations performed by d-shallow twists in an RT-sequence executed on the tree. Consider any d-shallow twist in the sequence. Define the *path* of the twist to be the path resulting from the contraction of the right edges on the access path of the lowest node involved in the twist. We classify the rotations performed by the twist as follows:

Type 1. Local, interblock rotation in which the top node is global: There is at most one type 1 rotation per twist because, the left subtree of the bottom node of the rotation consists of only local nodes. Note that type 1 rotations can actually occur: The lowest interblock rotation performed by the twist is type 1 if the top node of the rotation is global and the block of the bottom node does not contain any global nodes on the path of the twist.

Type 2. Local, interblock rotation in which the top node is local: The top node lies on the left path of its block and, since the node is local, it has $4d$ global ancestors within the block that lie on the path of the twist. Notice that the top nodes of different type 2 rotations belong to different blocks. Thus, if k_2 denotes the number of type 2 rotations performed by the twist, then the path of the twist contains at least $(4d+1)k_2$ edges. Since the number of edges on the path of the twist is bounded by dk, we obtain that $k_2 \leq \lfloor k/4 \rfloor$.

Type 3. Local, intrablock rotation: For each type 3 rotation, charge $(8/3)$ debits to the block in which the rotation is performed. If the number of type 3 rotations is at least $(3k-4)/8$, the total charges to the blocks *plus* a charge of $(4/3)$ debits to the twist itself pays for all the rotations performed by the twist.

Type 4. Global rotation: Only the situation where the number of type 3 rotations is less than $(3k-4)/8$ needs to be considered. In this case at least

$$k - 1 - \lfloor k/4 \rfloor - (\lceil (3k-4)/8 \rceil - 1) = k - \lfloor k/4 \rfloor - \lfloor (3k+3)/8 \rfloor \geq 3k/8$$

global rotations are performed. Therefore the global twist performs at least $3k/8$ rotations on the global tree. The global twist is $(8d/3)$-shallow because, the left depth of the lowest node of the global tree involved in the global twist is at most dk. If we charge each such global twist $(4/3)$ times the actual cost, then all the rotations can be paid for. This is seen as follows. Let k_3 and k_4, respectively, denote the number of type 3 and type 4 rotations. Then, $k_3 + k_4 \geq 3k/4 - 1$. The total charge is $8k_3/3 + 4/3 + 4k_4/3$ which is minimized when $k_3 = 0$. When $k_3 = 0$, the total charge is at least $4/3 + (4/3)(3k/4 - 1) \geq k$.

Since each d-shallow twist is charged at most $4/3$ debits, the total charge to all the d-shallow twists is at most $16dn/3$ by Lemma 11. The total charge to a block of size s is at most $8\binom{s}{2}/3$. It follows that the total charge to all the blocks is at most $4nK/3 \leq 116dn/3$. By the inductive hypothesis, the total charge to all the $(8d/3)$-shallow global twists is at most $(4/3)(87)(8d/3)(n/7.25) = 128dn/3$. Therefore the sum total of all the charges is bounded by $16dn/3 + 116dn/3 + 128dn/3 \leq 87dn$, completing the induction step. ■

6 A Conjecture

Consider a binary tree B. A *right ascent* of any node x in B is a consecutive series of right rotations of the edge connecting x and its parent. The conjecture:

Conjecture 6 (Turn-ascent conjecture) *The maximum number of right 2-turns in any intermixed sequence of right 2-turns and r right ascents performed on an n-node binary tree is $O(n + r)$.*

The turn conjecture implies the turn-ascent conjecture which in turn implies the deque conjecture. We show how to derive the deque conjecture from the turn-ascent conjecture. Referring to the proof of Theorem 7, we may conclude that the deque conjecture is true if the total cost of the operations performed on the left block in each epoch is linear in the size, say k, of the starting tree for the epoch. For any given epoch, the transformations of Lemma 10 can be applied to the sequence of left block operations performed during the epoch to construct a sequence of right 2-turns and $O(k)$ right ascents for a binary tree of size at most k, such that the total cost of all left block operations performed during the epoch is $O(k +$ the number of 2-turns in the new sequence). We obtain the deque conjecture by applying the turn-ascent conjecture to the new sequence.

Our bound of $\theta(n \log n)$ for $U_2(n)$ implies that the turn-ascent conjecture is true if $r = \Omega(n \log n)$. When $r = O(n \log n)$, our lower bound construction for $U_2(n)$ can be used to obtain sequences comprising $\Omega(n + r)$ right 2-turns. The difficulty of constructing sequences in which the number of right 2-turns is superlinear in the number of right ascents suggests that the conjecture is probably true.

Acknowledgements

My advisor Professor Richard Cole contributed considerably to this work. He exposed me to his work on the finger conjecture [1], from which I absorbed the ideas of blocks and global trees. Lemma 9 emerged during one of our discussions. He thoroughly read many earlier drafts of the paper and suggested several improvements. I am indebted to Professor Bud Mishra for comments on an earlier draft and encouragement.

References

1. R.Cole, B.Mishra, J.Schmidt, and A.Siegel, A proof of a partial form of the dynamic finger conjecture for splay trees, in preparation.

2. K.Culik II and D.Wood, A note on some tree similarity measures, *Info. Proc. Lett.* 15 (1982), 39-42.

3. S.Hart and M.Sharir, Nonlinearity of Davenport-Shinzel sequences and of generalized path compression schemes, *Combintorica* 6 (2) (1986), 151-177.

4. J.M.Lucas, Arbitrary splitting in splay trees, Rutgers University Tech. Rept. DCS-TR-234, June 1988.

5. D.D.Sleator and R.E.Tarjan, Self-adjusting binary search trees, *J. ACM* 32 (1985), 652-686.

6. D.D.Sleator, R.E.Tarjan, and W.P.Thurston, Rotation distance, Triangulations, and Hyperbolic Geometry, *Proc. 18th ACM STOC* (1986), 122-135.

7. R.E.Tarjan, Sequential access in splay trees takes linear time, *Combinatorica* 5 (1985), 367-378.

8. R.Wilber, Lower bounds for accessing binary search trees with rotations, *Proc. 27th IEEE FOCS* (1986), 61-70.

NYU COMPSCI TR-427
Sundar, Rajamani
Twists, turns, cascades,
deque conjecture, and
scanning theorem. c.2

NYU COMPSCI TR-427
Sundar, Rajamani
Twists, turns, cascades,
deque conjecture, and
scanning theorem. c.2

DATE DUE	BORROWER'S NAME

This book may be kept

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

GAYLORD 142 PRINTED IN U.S.A.